Pintos Project 4: File Systems

03/01/24

Outline

— Motivation

— Project Requirements
- Buffer Cache
- Indexed and Extensible Files
- Subdirectories
- Synchronization

— Suggested Order of Implementation

— Tips

Motivation

Pintos has a very limited file system implementation.

- No support for subdirectories
- Files cannot grow after they are created
- Each file must be allocated in one contiguous space
- Requires external synchronization

In project 4, you will remove these limitations.

Getting Started

- Build on top of project 2 or project 3
- Using project 3 code can get you up to 5% extra credit
  - To do so, enable VM in `src/filesys/Make.vars`

Project Components

Buffer Cache

When a request is made to read or write to a block:

1. Attempt to retrieve data from the cache
2. If not present, fetch the block from disk into the cache

All disk accesses will go through the buffer cache.

Cache size should be no greater than 64 sectors.

- If the cache becomes full, we must perform **eviction**.
- The replacement algorithm for this must be at least as good as the clock algorithm (as measured by disk accesses).

Buffer Cache

write-behind

Keep dirty blocks in the cache, rather than immediately writing to disk. They will only be written to disk in the following cases:

- When they are evicted
- Periodic flush to disk (using `timer_sleep` or otherwise)
- When Pintos halts (in `filesys_done()`)

read-ahead

If one block of a file is read, fetch the next block into the cache. Fetching the second block should be handled asynchronously, in the background.

Indexed & Extensible Files

Currently files are stored in contiguous memory.

This leads to external fragmentation: we cannot satisfy an allocation request of size n unless we have a single contiguous space of size n, even if enough aggregate disk space is available.

Indexed & Extensible Files

In project 4, you will modify `struct inode_disk` to eliminate this problem. This means using multi-level indexing: direct, indirect, and doubly indirect blocks.

If implemented correctly, this will allow your file system to support files as large as 8MB.

Indexed & **Extensible** Files

File system must support file growth

- Files start with size zero
- Writing past EOF expands file to specified position — bytes between old EOF and new write are zeroed out
- Reading from beyond the EOF returns no bytes

Subdirectories

Implement hierarchical namespace (e.g. `src/filesys/foobar.txt`).

- Only have to support 14-character file names, but must allow much longer full path names.
- Track the current directory of a process (set to root at startup).
- Ensure child processes inherit the current directory of the parent.

Path resolution requirements

- Absolute and relative paths
- “.” and “..”

Subdirectories

Update system calls to support directories

- `open()` can open directories
- `close()` can close directories
- `remove()` can delete empty directories except root

Implement new system calls: `chdir`, `mkdir`, `readdir`, `isdir`, `inumber`

Synchronization

No more global file system lock!

Operations on independent entities should not wait on one another.

Synchronization — Cache Blocks

- Operations on different cache blocks must be independent.
- When I/O is required on one block, operations on other blocks that do not require I/O should proceed without waiting for I/O to complete.

Synchronization – Single File

- Concurrent reads can complete without waiting for one another.
- Concurrent writes can occur, as long as the file is not being extended.
- Extending a file and writing data to the new section must be atomic.

Synchronization — Directories

- Operations on different directories should take place concurrently
- Operations on the same directory can wait for one another.
- Note that each `struct file` and `struct dir` object is only accessed by a single thread.

Suggested Order of Implementation

1. Buffer cache
2. Extensible files
3. Subdirectories
4. Remaining items

Tips

1. Start early (as usual)
2. Implement the buffer cache at the start
3. Make sure you understand the synchronization requirements before you implement each component